Seminar 7

Pentru fiecare dintre secventele de cod de mai jos, spuneti daca secventa compileaza sau nu. In caz afirmativ, spuneti ce va afisa (in cazul in care standardul C++ spune ca e comportament nedefinit, puteti specifica acest lucru). In caz negativ, sugerati o modificare, prin editarea a cel mult o linie de cod (modificarea unei linii de cod, adaugarea unei linii de cod sau stergerea unei linii de cod), care sa faca secventa sa compileze si spuneti ce afiseaza noua secventa de cod.

Secventa 1

```
#include<iostream>
struct Integer {
    int x;
    Integer(const int val = 0) : x(val){}
    friend Integer operator+ (Integer& i, Integer& j) {
        return Integer(j.x + i.x);
    }
    friend std::ostream& operator<<(std::ostream& o, Integer i) {
        o << i.x; return o;
    }
};
int main () {
    Integer i(25), j(5), k(2020);
    std::cout << (i + j + k);
}
```

Rezultat:

```
main.cpp:16:25: Invalid operands to binary expression ('Integer' and 'Integer')
```

Secventa 2

```
#include <iostream>
#include <string>
struct A {
    A (int i=0) {std::cout<<"A" << i;}
    ~A () {std::cout<<"~A";}
};
struct B : public A {
    A a;
    B() : a(25) {std::cout<<"B";}
    virtual ~B() {std::cout<<"~B";}
};
struct C: public B {
    C () {std::cout << "C";}
    ~C (){std::cout << "~C";}
};

int main () {
    A *a = new C();
    delete a;
}
```

Rezultat:

```
A0A25BC~A
```

Secventa 3

```
#include <iostream>
struct A {
    virtual void foo () {}
};
struct B : public A {
    void foo () {};
};
class D: public B {
public:
    void foo () {};
    std::string bar() {return "bar";}
};
int main () {
    A *p = new D();
    if(dynamic_cast<B*>(p)) {std::cout << "Type is B";}
    else if (dynamic_cast<D*>(p)) {std::cout << "Type is D";}
    else { std::cout << "Conversion failed";}
}
```

Rezultat:

```
Type is B
```

Secventa 4

```
#include <iostream>
struct A {
    virtual void foo () {std::cout<<"A";}
};
struct B : public A {
    void foo () {std::cout<<"B";};
};
class D: public B {
public:
    void foo () {std::cout<<"D";};
};
int main () {
    A *pa[] = {new B, new D, new B, new D, new B, new D};
    for (int i = 0; i < 6; i++) {
        if (i % 2 == 0) {
            D d = dynamic_cast<D&>(*pa[i]);
            d.foo();
        } else {
            B b = dynamic_cast<B&>(*pa[i]);
            b.foo();
        }
    }
}
```

Rezultat:

```
terminating with uncaught exception of type std::bad_cast: std::bad_cast
```

Secventa 5

```
#include <iostream>
class C {
protected:
    int x;
public:
    C(int y): x(y) {}
    virtual C operator+(const C& c) const {
        return C(this->x + c.x);
    }
    friend std::ostream& operator << (std::ostream& o, C c) {
        o << c.x; return o;
    }
};

class D: public C {
public:
    D(int y) : C(y) {}
    C operator+(const C& c) const {
        return C(x + 22);
    }
};

int main () {
    C *c = new D(4);
    std::cout << *c + C(2);
}
```

Rezultat:

```
26
```

Secventa 6

```
#include <iostream>

int foo (int x, int y = 0) {
    return x + y - 2020;
}

int foo (int x) {
    return x + 2020;
}

int main () {
    std::cout << foo(5);
}
```

Rezultat:

```
Nu compileaza: line 12 foo is ambiguous
```

Secventa 7

```
#include <iostream>
struct A {
    A (int i=0) {std::cout<<"A" << i;}
    ~A () {std::cout<<"~A";}
};
struct B : public A {
    A a;
    B() : a(25) {std::cout<<"B";}
    ~B() {std::cout<<"~B";}
};
struct C: public A, public B {
    C () {std::cout << "C";}
    ~C (){std::cout << "~C";}
};

int main () {
    C c;
}
```

Rezultat:

```
A0A0A25BC~C~B~A~A~A
```

Secventa 8

```
#include <iostream>
#include <string>
struct A {
    A () {std::cout<<"A";}
    ~A () {std::cout<<"~A";}
};
struct B : public A {
    A a;
    B() {std::cout<<"B";}
    ~B() {std::cout<<"~B";}
};
struct C: public A, public B {
    C () {std::cout << "C";}
    ~C (){std::cout << "~C";}
};

int main () {
    A *c = new C();
}
```

Rezultat:

```
main.cpp:18:12 Ambiguous conversion from derived class 'C' to base class 'A':
struct C -> struct A
struct C -> struct B -> struct A
```

Secventa 9

```cpp
#include<iostream>

struct Integer {
    int x;
    Integer(const int val = 0) : x(val){}
    Integer operator+ (Integer& i) {
        return Integer(x + i.x);
    }
    friend std::ostream& operator<<(std::ostream& o, Integer& i) {
        o << i.x; return o;
    }
};

int main () {
    Integer i(25), j(5), k(2020);
    std::cout << (i + j + k);
}
```

Rezultat:

```
main.cpp:16:15: Invalid operands to binary expression ('std::__1::ostream' (aka '
    basic_ostream<char>') and 'Integer')
```

Secventa 10

```
#include <iostream>
struct A {
    virtual void foo () {}
};
struct B : public A {
    void foo () {};
};
struct D: public B {
    void foo () {};
    std::string bar() {return "bar";}
};
int main () {
    A *p = new B();
    D *pd = dynamic_cast<D*>(p);
    if (pd != nullptr) { std::cout << "D"; }
    else { std::cout << "incompatible"; }
    std::cout<<pd->bar();
}
```

Rezultat:

```
incompatiblebar
```

Secventa 11

.

```
#include<iostream>
#include<climits>
struct Array {
    int *x, size;
    Array(const int& val = 5) : x(new int[val]), size(val){
        for (int i = 0; i < size; i++) *(this->x+i) = i;
    }
    int operator[](unsigned i = 0) const {
        return i > size ? INT_MAX : x[i];
    }
};
int main () {
    Array a(2020);
    for (unsigned i = 0; i < a.size; i++) {
        std::cout << a[i] << " ";
    }
}
```

Rezultat:

```
main.cpp:9:29: Parameter of overloaded 'operator[]' cannot have a default argument
```

Secventa 12

```
#include<iostream>
class B {
protected:
    int x;
public:
    B (int y = 2020) : x(y) {}
};

class D : public B {
public:
    D(int y) : B(y) {}
    D operator+ (const B& b) {
        return D(x + b.x);
    }
    operator int () const {return x;}
};

int main () {
    D d = (D(22) + D(5));
    std::cout << d;
}
```

Rezultat:

```
main.cpp:15:24: 'x' is a protected member of 'B'
```

Secventa 13

```
#include<iostream>
struct B {
    virtual void foo () {std::cout<<"B";}
};
struct C: public B {
    void foo () {std::cout<<"C";}
};
struct D: private B {
    void foo () {std::cout<<"D";}
};
void call (B& b) {
    b.foo();
}
int main () {
    C c; D d;
    call(c); call(d);
}
```

Rezultat:

```
main.cpp:18:19: Cannot cast 'D' to its private base class 'B'
```